Apple Assembly Line

$1.50

Volume 3 -- Issue 2 November, 1982

In This Issue...

Apple/Fest in Houston

Although only about one-third the size of the Boston original (last May), it was still worth the trip. I met an orthopedic surgeon from Lille, France, who flew down Saturday from New York just for the show. Also a professor from Des Moines. There was not a lot to see that could be called NEW, but it was valuable to meet and get to know the people. I brought along my son David, almost ten now; he loved playing all the new games, and was a big help in the IAC booth.

If you are in an area where there is no club of Apple owners, you might like to contact the International Apple Core at 908 George St, Santa Clara, CA 95050. They have a start-up kit for new clubs that will help you organize your own club.

Bill Morgan also came on Saturday. We have kidded Bill in the past that he looked a lot like Paul Lutus...well, three people were almost positive on Saturday!

Another Christmas Special

And this one is in December! Subscribers have until the end of 1982 to get Laumer Research's FLASH Integer Basic Compiler at only $49. That is a savings of nearly 38%!

A Sight of Sound.....Herbert A. & Herbert L. McKinstry

The Apple-Talker program that came on our disk for the S-C Assembler II Version 3.2 does some interesting things that go beyond what it was designed to do. When we tried it out we played a recorded message from our cassette recorder into the Apple memory and were amazed at the computer rendering of the original words.

The actual quality of reproduction leaves something to be desired, so when someone said "Let's see what it sounds like," and another said "Let's hear what it looks like," we snooped around the program listing and found that what we were hearing was stored on Hi-Res graphics page one. We looked at it by typing in $C050:0 and $C057:0. The sight of the sound was not too loud, nor was it even obvious that what we were looking at was the sound that we had heard.

If there were a pitch there, we should see some kind of pattern. We recorded a pitch, and saw that the sound was noisy. So then we entered some sense into memory by creating a repeating pattern, and listened to the patterns. We tried some like this:

```
*2000:FF FF 00 00 N 2004<2000.27FFM
*2800:FF 00 N 2802<2800.2FFFM
*3000:F0 N 3001<3000.37FFM
*3800:CC N 3801<3800.3FFEM
```

and

```
*2000:FC 0F C0 00 N 2004<2000.27FFM
*2800:F8 3F 03 E0 N 2804<2800.2FFFM
*3000:AA N 3001<3000.37FFM
*3800:CC N 3801<3800.3FFEM
```

We liked what we saw and we saw what we heard, so our thanks to Bob Sander-Cederlof and to Victor Borge for his recognition of the sight of sound.

Your Apple Can Talk.....Bob Sander-Cederlof

Back in the summer of 1978, I spent two weeks in California with my kids. I visited a couple of computer stores with my brother, to show him what my Apple looked like. In one of them, I think the Byte Shop in Westminster on Beach Blvd., the proprietor mentioned in passing an astonishing event. He told me, "A high schooler was in here a few weeks ago with a program that produced speech out of the Apple speaker!" "Impossible," I mumbled.

A few weeks later I heard rumors of a program by Bob Bishop which did indeed make the Apple talk. I think it was in the September meeting of the Dallas Apple Corps that I overheard his program running. From amazement to insight took only a few seconds...I almost RAN home to write a program to do the same thing!

S-C Macro Assembler (the best there is!).....$80.00
S-C Macro Cross Assembler Modules
6800/6801/6802 Version.....$32.50
6809 Version.....$32.50
Z-80 Version.....$32.50
68000 Version.....$50.00
Requires ownership of S-C Macro Assembler.
Each disk includes regular and language card versions.
Upgrade from Version 4.0 to MACRO.....$27.50
Source code of Version 4.0 on disk.....$95.00
Fully commented, easy to understand and modify to your own tastes.

ES-CAPE: Extended S-C Applesoft Program Editor.....$60.00

AAL Quarterly Disks.....each $15.00
Each disk contains all the source code from three issues of "Apple Assembly Line", to save you lots of typing and testing time.
QD#1: Oct-Dec 1980 QD#2: Jan-Mar 1981 QD#3: Apr-Jun 1981
QD#4: Jul-Sep 1981 QD#5: Oct-Dec 1981 QD#6: Jan-Mar 1982
QD#7: Apr-Jun 1982 QD#8: Jul-Sep 1982 QD#9: Oct-Dec 1982

Double Precision Floating Point for Applesoft.....$50.00
Provides 21-digit precision for Applesoft programs.
Includes sample Applesoft subroutines for standard math functions.

FLASH! Integer BASIC Compiler (Laumer Research).....(regular $79) $49.00
Special price to AAL readers only, until 12/31/82!
Source Code for FLASH! Runtime Package.....$39.00

Super Disk Copy III (Sensible Software).....(reg. $30.00) $27.00
Amper-Magic (Anthro-Digital).....(reg. $75.00) $67.50
Quick-Trace (Anthro-Digital).....(reg. $50.00) $45.00
Cross-Reference and Dis-Assembler (Rak-Ware).....$45.00
Apple White Line Trace (Lone Star Industrial Computing).....$50.00
(A unique learning tool)

Blank Diskettes (with hub rings).....package of 20 for $50.00
Small 3-ring binder with 10 vinyl disk pages and disks.....$36.00
Vinyl disk pages, 6"x8.5", hold one disk each.....10 for $4.50
Reload your own NEC PC-8023 ribbon cartridges.....each ribbon $5.00
(Messy, but effective!)
Diskette Mailing Protectors.....10-99: 40 cents each
100 or more: 25 cents each
Corrugated folder specially designed for mailing mini-floppy diskettes. Fits in standard 6x9-inch envelope. (Envelopes 5-cents each, if you need them.)

Ashby Shift-Key Mod.....$15.00
Paymar Lower-Case Adapter.....$37.50
For Apples before Revision 7 only
Lower-Case Display Encoder ROM.....$25.00
Works only Revision level 7 Apples. Replaces the encoder ROM.

Books, Books, Books.....compare our discount prices!
"Beneath Apple DOS", Worth & Lechner.....($19.95) $18.00
"Bag of Tricks", Worth & Lechner, with diskette.....($39.95) $36.00
"Apple Graphics & Arcade Game Design", Stanton.....($19.95) $18.00
"Assembly Lines: The Book", Roger Wagner.....($19.95) $18.00
"What's Where in the Apple", Second Edition.....($24.95) $23.00
"6502 Assembly Language Programming", Leventhal.....($16.99) $16.00
"6502 Subroutines", Leventhal.....($12.99) $12.00
"MICRO on the Apple--1", includes diskette.....($24.95) $23.00
"MICRO on the Apple--2", includes diskette.....($24.95) $23.00
"MICRO on the Apple--3", includes diskette.....($24.95) $23.00

*** S-C SOFTWARE, P. O. BOX 280300, Dallas, TX 75228 ***
*** (214) 324-2050 ***
*** We take Master Charge, VISA and American Express ***

A month or two later I handed out copies of the program and gave a talk on the subject of speech synthesis and recognition. When Version 3.2 of the S-C Assembler was released, I included the same program as an example. Then again on version 4.0

Meanwhile, Bob Bishop released several neat tapes through Softape, including a talking calculator, "Apple Talker", and "Apple Listener". The latter program did some limited speech recognition through the cassette port, with no additional hardware. I bought the last two, and I still have the tape somewhere, but I have never loaded it.

About two years later Muse Software started marketing a program on disk to evoke speech from the Apple. I believe they included some sort of "editor" to allow you to make your own programs talk. I never saw or heard it, so I don't know.

As far as I know, the basic idea behind all of these programs is the same: approximate the waveform of spoken words by toggling the Apple speaker. You can hook a microphone up to the cassette input port and toggle the output speaker whenever the input port changes. Or you can record a message on tape, and "play" in into the Apple.

My program samples the cassette input port about 6000 times per second. If the input byte is $80 or larger, I store a "1"; if less than $80, I store a "0". I pack eight bits in a byte, and store the bytes in a buffer from $4000 through $5FFF. The buffer is 8192 bytes long, so that is 65536 samples or about 10 seconds of stored sound. You could store more samples or less samples, according to your own needs.

The playback loop looks at the stored bits at the same rate, and toggles the speaker whenever there is a change from 1 to 0 or 0 to 1. The result is actually understandable, though somewhat scratchy.

As the McKinstry's pointed out, my choice of buffer coincides with the Hi-Res Graphics page. In the copy of the program they have, I used $2000-3FFF, which is Hi-Res page one. Now I use $4000-5FFF, Hi-Res page two, so it will not erase the last half of the S-C Assembler when I test the program. Taking their suggestions to heart, I added the code to turn on the Hi-Res display during recording and playback, and to turn it off when finished.

When looking at the display you need to bear in mind the complex way the bytes are arranged on the Hi-Res screen. For starters, the bits are backwards in each byte. And remember that only seven bits of each byte show up on the screen -- the 8th bit shifts the other seven one half dot position. And the big confuser is the way the lines are arranged. (See Mike Laumer's article in the July ,1982 issue of AAL, page 15ff, for a discussion of the line arrangement.)

I prepared some EXEC files which initialize the buffer to various patterns, including the ones the two Herberts

S&H Software is Speeding up The Apple®!

The DOS Enhancer (TDE) works up to 500% faster than standard Apple DOS 3.3...$69.95

TIRED OF WAITING . . . for your programs to load or save? Then S&H's TDE, licensed by Apple, is the answer.

TDE program updates standard Apple DOS 3.3 disks — or creates copyable TDE disks — with TDE's QuickDOS and Quick-load features.

TDE's QuickDOS runs and saves BASIC and binary programs up to 5 times faster* and is completely compatible with standard Apple DOS 3.3 programs.

TDE "Quick-loads" the RAM card with FPBASIC/INTBASIC, QuickDOS or user program in 1.7 seconds at startup.

TDE "package" includes utility disk, training/support disk, step-by-step instruction manual, S&H's Supercat/menu, and multidrive "Quick-copy" program.

TDE system requirements: 48K Apple][or][+ ROM/RAM card, DOS 3.3 and one or more disk drives.

Here's what the critics say:

● John Mitchener of PEELINGS II: "The speed increase with **TDE** is awesome and is probably worth the price of the program alone, without all the other features. AA rating."

● Val Golding, Editor of Call-A.P.P.L.E.: "**(TDE)** stands as a shining example of how utility and application programs take into account every possible system configuration."

● Clark Congleton of The Apple Orchard: "The Quick-load capabilities will make this package attractive to anyone who spends a lot of time at the keyboard."

● Chuck Carpenter of INFOWORLD: "Results in a disk that will boot — very fast — in any Apple system."

*To achieve speeds even faster than a hard-disk drive, combine **TDE** with Axlon's 320K Memory System.

Amper-Sort/Merge (A-S/M II) works up to 1000% faster than even VisiCorp's VisiFile program...$69.95

A-S/M II is the fastest "file clerk" you've ever met. Of all the sort utilities developed to manage Apple][data files none does the job nearly so fast as A-S/M II!

A-S/M II's new features include: S&H's superfast VisiFile index sort (callable from within VisiFile for effortless use), an equally fast S&H random access file index sort, and parameter file editing.

A-S/M II can sort/merge from one to five unsorted files into a single file of up to 125K in size per disk.

A-S/M II's "package" includes utility disk, training disk, step-by-step instruction manual, and S&H's new Supercat/menu.

A-S/M II's system requirements: 48K Apple][with ROM or RAM card or 48K Apple][+ with DOS 3.3 and Disk II.

Dealer inquiries invited

Apple is a registered trademark of Apple Computer, Inc.

S&H Software

Available from your dealer. **Mail Order:** Send checks to S&H Software, 58 Van Orden Rd., Harrington Park, NJ 07640. 201-768-3144

Credit Cards: Phone Cybertronics International at 212-532-3089. (Overseas Airmail: Add $3.00 postage and handling.)

suggested. (It is amazing how handy it is to be able to create/modify little text files like these using the editor in the S-C Macro Assembler!)

```
Sound #1

$4000:FF FF 00 00 N 4004<4000.47FFM
$4800:FF 00 N 4802<4800.4FFFM
$5000:F0 N 5001<5000.57FFM
$5800:CC N 5801<5800.5FFEM


Sound #2

$5800:CC N 5801<5800.5FFEM
$5000:AA N 5001<5000.57FEM
$4800:F8 3F 03 E0 N 4804<4800.4FFCM
$4000:FC 0F C0 00 N 4004<4000.47FCM


Sound #3

$4000:00 01 03 07 0F 1F 3F 7F FF FE FC F8 F0 E0 C0 80
$4010<4000.5FEFM


Sound #4

$4000:00 FF 00 00 FF FF 00 00 00 00 FF FF FF FF
$400E<4000.5FFFM


Sound #5

$4000:00 88 00 00 88 88 00 00 00 00 88 88 88 88
$400E<4000.5FFFM
```

To play back one of the sound above, simple EXEC or type in the monitor commands, and then "MGO TALK".

Looking at the program which follows, you find three main routines. ECHO (lines 1180-1300) samples the cassette port about 6000 times per second; if it has changed, the speaker is toggled. After each toggle the keyboard strobe is examined, so that typing any key can stop the program and return to the caller.

RECORD (lines 1560-1710) stores 65536 samples in the buffer. TALK (lines 1750-1950) play back the buffer contents. You can play with the sample rate and playback rate by modifying the constant 30 in lines 1590 and 1790. It is amusing to play back a message faster or slower than it was recorded.

Both RECORD and TALK use a monitor subroutine called NXTA to control the loop. This is the same subroutine used by the monitor memory display and memory move commands. NXTA tests the current value of A1L,A1H ($3C,$3D) against A2L,A2H ($3E,$3F), and sets carry if A1 is greater than or equal to A2.

Then it increments A1.

I tried various schemes for packing the bits in the buffer, to save space for more speech. None of them were effective enough to bother with, but you might run on to one that is. I also experimented with isolating words and individual phonemes, and with trying to filter out the scratchiness. I was not satisfied with any of my results. If you are successful, I would like to hear about it.

[A later note: I just received Dec-82 Creative Computing, and there are reviews of several speech synthesis systems. One, called "Software Automatic Mouth (SAM)", is claimed to be a "high quality speech synthesizer created entirely in software." SAM costs $125 ($99 from Huntington Computing from now until the end of the year). In spite of the claim, it is not entirely software. There is also a small board containing a digital-to-analog converter (DAC), an audio amplifier, and a volume control. You can hook it up to the speaker in the Apple, or supply an external speaker. The ad claims it enables you to add speech to your programs with ease, but bear in mind that the software takes 9K of RAM, and 6K more if you want to automatically translate straight English text to speech.]

```
                    1000 *--------------------------------
                    1010 *       APPLE-TALKER FROM S-C SOFTWARE CORP.
                    1020 *--------------------------------
FCBA-               1030 MON.NXTA .EQ $FCBA      BUMP AND TEST A1
                    1040 *--------------------------------
C060-               1050 CASSETTE .EQ $C060      CASSETTE INPUT LEVEL
C030-               1060 SPEAKER  .EQ $C030      SPEAKER OUTPUT
C010-               1070 STROBE   .EQ $C010
C000-               1080 KEYBOARD .EQ $C000
                    1090 *--------------------------------
002F-               1100 LAST     .EQ $2F        LAST CASSETTE INPUT LEVEL
003C-               1110 A1L      .EQ $3C        MONITOR A1L, A1H, A2L, A2H
                    1120 *--------------------------------
0800- 00 40         1130 BUFFER   .DA $4000      FWA OF BUFFER
0802- FF 5F         1140          .DA $5FFF      LWA OF BUFFER
                    1150 *--------------------------------
                    1160 *    ECHO CASSETTE THRU SPEAKER
                    1170 *--------------------------------
0804- A0 1E         1180 ECHO     LDY #30        150 USEC DELAY
0806- 88            1190 .1       DEY
0807- D0 FD         1200          BNE .1
0809- AD 60 C0      1210          LDA CASSETTE
080C- 45 2F         1220          EOR LAST       SEE IF TOGGLED
080E- 10 F4         1230          BPL ECHO       NO
0810- 45 2F         1240          EOR LAST       YES
0812- 85 2F         1250          STA LAST
0814- AD 30 C0      1260          LDA SPEAKER    TOGGLE SPEAKER
0817- AD 00 C0      1270          LDA KEYBOARD
081A- 10 E8         1280          BPL ECHO
081C- 8D 10 C0      1290          STA STROBE
081F- 60            1300          RTS
                    1310 *--------------------------------
                    1320 *    SET UP BUFFER ADDRESSES
                    1330 *--------------------------------
0820- A2 03         1340 SETUP    LDX #3
0822- BD 00 08      1350 .1       LDA BUFFER,X
0825- 95 3C         1360          STA A1L,X
0827- CA            1370          DEX
0828- 10 F8         1380          BPL .1
082A- 86 2F         1390          STX LAST
082C- AD 50 C0      1400          LDA $C050    SELECT HGR2 FOR VIEWING
082F- AD 52 C0      1410          LDA $C052
0832- AD 55 C0      1420          LDA $C055
0835- AD 57 C0      1430          LDA $C057
0838- 60            1440          RTS
                    1450 *--------------------------------
                    1460 *    RESTORE NORMAL SCREEN AND EXIT
                    1470 *--------------------------------
0839- AD 51 C0      1480 FINISH   LDA $C051
083C- AD 53 C0      1490          LDA $C053
083F- AD 54 C0      1500          LDA $C054
0842- AD 56 C0      1510          LDA $C056
0845- 60            1520          RTS
                    1530 *--------------------------------
                    1540 *    STORE SPEECH IN BUFFER
                    1550 *--------------------------------
0846- 20 20 08      1560 RECORD   JSR SETUP      SET UP BUFFER ADDRESSES
0849- A2 08         1570 .1       LDX #8         EIGHT BITS
084B- 48            1580 .2       PHA            PUSH BYTE WE ARE FILLING
084C- A0 1E         1590          LDY #30
084E- 88            1600 .3       DEY            150 USEC DELAY
084F- D0 FD         1610          BNE .3
0851- AD 60 C0      1620          LDA CASSETTE   READ CASSETTE LEVEL
0854- 0A            1630          ASL            LEVEL INTO CARRY BIT
0855- 68            1640          PLA
0856- 2A            1650          ROL            MERGE LEVEL INTO BYTE
0857- CA            1660          DEX
0858- D0 F1         1670          BNE .2         BYTE NOT FULL YET
085A- 81 3C         1680          STA (A1L,X)    STORE NEXT WORD IN BUFFER
085C- 20 BA FC      1690          JSR MON.NXTA       BUMP & TEST POINTER
085F- 90 E8         1700          BCC .1         NOT THRU
0861- 4C 39 08      1710          JMP FINISH
```

```
                        1720 *--------------------------------
                        1730 *    PLAYBACK SPEECH FROM BUFFER
                        1740 *--------------------------------
0864- 20 20 08  1750 TALK   JSR SETUP      SET UP BUFFER ADDRESSES
0867- A2 00     1760 .1     LDX #0
0869- A1 3C     1770        LDA (A1L,X)    GET NEXT WORD FROM BUFFER
086B- A2 08     1780        LDX #8         EIGHT BITS
086D- A0 1E     1790 .2     LDY #30
086F- 88        1800 .3     DEY            150 USEC DELAY
0870- D0 FD     1810        BNE .3
0872- 45 2F     1820        EOR LAST       TEST IF LEVEL CHANGED
0874- 10 13     1830        BPL .5         NO
0876- 45 2F     1840        EOR LAST       YES, RESTORE (A)
0878- 85 2F     1850        STA LAST       UPDATE LEVEL
087A- AC 30 C0  1860        LDY SPEAKER    TOGGLE SPEAKER
087D- 0A        1870 .4     ASL
087E- CA        1880        DEX
087F- D0 EC     1890        BNE .2
0881- 20 BA FC  1900        JSR MON.NXTA       BUMP & TEST POINTER
0884- 90 E1     1910        BCC .1         NOT THRU
0886- 4C 39 08  1920        JMP FINISH
0889- 45 2F     1930 .5     EOR LAST       RESTORE (A)
088B- 4C 8E 08  1940        JMP .6         EVEN OUT TIMING
088E- 4C 7D 08  1950 .6     JMP .4
```

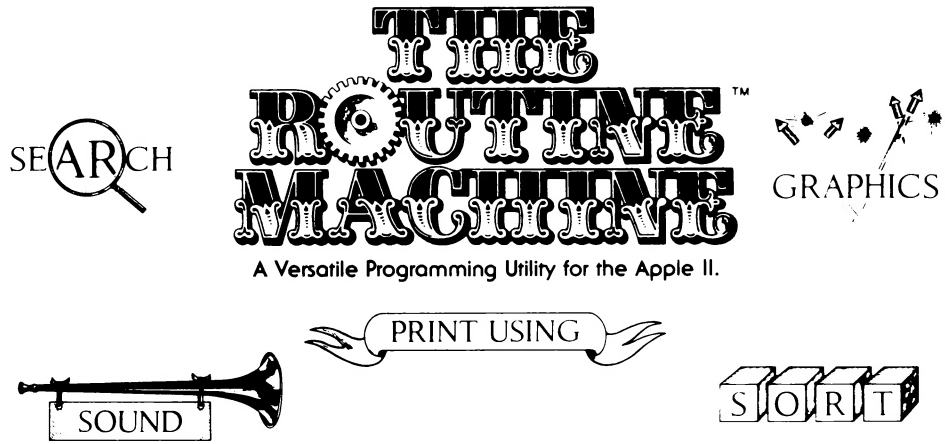
Speaking of Speech.....Bill Morgan

Just thought I'd tell you a little about the way I played around with a speech program like Bob's. I couldn't find the disk with the exact code, but here's what I remember about it. I wanted a brief Applesoft program which would say the numbers 0 through 9 when a number key was pressed.

To do this, first record your voice on tape, reciting the ten numbers. Then play the tape into your Apple, using the RECORD routine in Bob's program. Now, by using the system monitor to examine memory, it's easy to scan through the buffer and see where each word begins and ends. The gaps between words will be long stretches of "00 ... 00", with a few stray bytes of noise along the way. Words will be stretches of random-looking values. It's interesting to see the difference between a word like "two", which starts abruptly and trails off, and one like "eight", which starts more slowly and ends suddenly.

Now you can use the monitor move command to remove the gaps between words. Move the data for "one" to the very beginning of the buffer, and note its start and end addresses. Then move "two" down to the space just after "one", and note the addresses. Carrying on like this, you can compress the number data into about half the space of the original recording.

Assemble the playback portion of Bob's program at $300. All you should need is lines 1760-1950 (plus the needed .EQ's), with an RTS substituted for the JMP FINISH at line 1920. To say a number, all your Applesoft program has to do is get the starting and ending addresses of a word from an array, POKE the addresses into locations 60-63, and CALL 768.



Now, from the programming experts at S.D.S., an easy-to-use way of putting the POWER and SPEED of machine language routines in YOUR OWN APPLESOFT PROGRAMS!

ROUTINE MACHINE does all the work for you — no knowledge of machine language programming, whatsoever, is required. Simply choose the routine you need from an ever-growing library, and Routine Machine will effortlessly put them in your program, and all done transparently! No need to be aware of or bother with BLOAD's, HIMEM:, etc.

Best of all, with just this starter package, you'll have the routines to put High Resolution **graphics** and **sound** in your programs immediately! Also included is a versatile **print using** module to banish the "decimal point demons" forever! To round out the package, we've also included powerful **search** and **sort** routines (for single dimension arrays; Search: 1000 elements in 1 second. Sort: 1000 elements in 90 seconds), and a number of other often-needed routines as well (30 routines in all).

OUR GUARANTEE

IF YOU DON'T SAVE MORE THAN THE PURCHASE PRICE OF 'ROUTINE MACHINE' IN YOUR OWN PROGRAMMING TIME IN THE FIRST 30 DAYS YOU OWN IT, SIMPLY RETURN IT FOR A COMPLETE REFUND, NO QUESTIONS ASKED!

Additional library disks titled **"Ampersoft Program Library"** are already available.

Some of the other routines in The Routine Machine (plus others not listed) are:

SWAP: Swaps two string or numeric values.

TEXT OUTPUT: Prints with no "word break" on screen.

STRING OUTPUT: Input any string, regardless of commas, etc.

ERR: Stack fix for Applesoft ONERR handling.

GOTO, GOSUB: Allows computed statements. Example: **GOTO X *5** or **GOSUB X *5.**

BLOAD: Load any binary file 5 times faster than normal. Hi-Res pictures load in under 2 seconds.

RESET HANDLER: Treats RESET with ONERR; or will RUN or reboot disk.

HI-RES ASCII: Character set for mixing text Hi-Res graphics.

TURTLE GRAPHICS: Versatile Hi-Res graphics routines for easy drawing of Hi-Res figures.

southwestern data systems™

P.O. BOX 582 • SANTEE, CALIFORNIA 92071 • TELEPHONE: 714/562-3670

Even Faster Primes.....Anthony Brightwell

Is this the last word on prime number generation?

I modified Charles Putney's program from the February issue, and cut the time from 330 milliseconds down to 183 milliseconds! Here is what I did:

* I sped up the zero-memory loop by putting more STA's within the loop.

* I removed the CLC from the main loop. After all, why CLC within the loop if you're looping on a BCC condition?

* I removed the LDA #$FF from the main loop. It was there to be sure a non-zero value gets stored in non-prime slots, but why LDA #$FF if the accumulator never contains $00 within the loop?

* I changed the way squares of primes are computed. Charlie did it using a quick 8-bit by 8-bit multiply. I took advantage of a little number theory, and shaved off some time.

The method I use for squaring may appear very round-about, but it actually is faster in this case. Look at the following table:

Odd #'s	square	neat formula
1	1	0 * 8 + 1
3	9	1 * 8 + 1
5	25	3 * 8 + 1
7	49	6 * 8 + 1
9	81	10 * 8 + 1

The high byte of the changing factor in the "neat formula" is stored in the LDA instruction at line 1550, and the low byte in the ADC instruction at line 1900. The factor is the sum of the numbers from 1 to n: 1+2=3, 1+2+3=6, 1+2+3+4=10, etc. In all, 31 primes are squared, and the total time for all the squaring is less than 3 milliseconds.

Here is a driver in Applesoft to load the program and then print out primes from the data array.

```
10  REM DRIVER FOR TONY'S FAST PRIME FINDER
20  PRINT CHR$ (4)"BLOAD B.TONY'S SUPER-FAST PRIMES"
30  HOME : PRINT "HIT ANY KEY TO START"
40  GET A$: PRINT " GENERATING PRIMES     ."
50  CALL 32768
60  FOR A = 8195 TO 24576 STEP 2
70  IF  PEEK (A) = 0 THEN  PRINT A - 8192;" ";
80  NEXT
```

A few more cycles can probably still be shaved.... Any takers?

```
                        1010         .OR $8000     SAFELY OUT OF WAY
                        1020         .TF B.TONY'S SUPER-FAST PRIMES
                        1030 *-------------------------------
2000-                   1040 BASE    .EQ $2000     BASE OF PRIME ARRAY
FF3A-                   1050 BEEP    .EQ $FF3A     BEEP THE SPEAKER
                        1060 *-------------------------------
                        1070         .MA ZERO
                        1080         STA ]1+$001,X
                        1090         STA ]1+$101,X
                        1100         STA ]1+$201,X
                        1110         STA ]1+$301,X
                        1120         STA ]1+$401,X
                        1130         STA ]1+$501,X
                        1140         STA ]1+$601,X
                        1150         STA ]1+$701,X
                        1160         .DO ]1<$5800
                        1170         >ZERO ]1+$800
                        1180         .FIN
                        1190         .EM
                        1200 *-------------------------------
                        1210 *       MAIN CALLING ROUTINE
                        1220 *
8000- A9 64             1230 MAIN    LDA #100      DO 100 TIMES SO WE CAN MEASURE
8002- 8D 48 81          1240         STA COUNT     THE TIME IT TAKES
8005- 20 3A FF          1250         JSR BEEP      ANNOUNCE START
8008- 20 13 80          1260 .1      JSR PRIME
800B- CE 48 81          1270         DEC COUNT     CHECK COUNT
800E- D0 F8             1280         BNE .1        DONE ?
8010- 4C 3A FF          1290         JMP BEEP      SAY WE'RE DONE
                        1300 *-------------------------------
                        1310 *       PRIME ROUTINE
                        1320 *       SETS ARRAY STARTING AT BASE
                        1330 *       TO $FF IF NUMBER IS NOT PRIME
                        1340 *       CHECKS ONLY ODD NUMBERS > 3
                        1350 *       INC = INCREMENT OF KNOCKOUT
                        1360 *       N = KNOCKOUT VARIABLE
                        1370 *-------------------------------
                        1380 PRIME
8013- A2 01             1390         LDX #1
8015- 8E 30 81          1400         STX SHCNT+1   STARTING MULTIPLIER FOR SQUARE
8018- 8E 33 81          1410         STX MULT+1
801B- CA                1420         DEX
801C- 8E F1 80          1430         STX SQUARE+1
801F- 8A                1440         TXA           CLEAR WORKING ARRAY
8020-                   1450 .1      >ZERO BASE
8020- 9D 01 20          0000>         STA BASE+$001,X
8023- 9D 01 21          0000>         STA BASE+$101,X
8026- 9D 01 22          0000>         STA BASE+$201,X
8029- 9D 01 23          0000>         STA BASE+$301,X
802C- 9D 01 24          0000>         STA BASE+$401,X
802F- 9D 01 25          0000>         STA BASE+$501,X
8032- 9D 01 26          0000>         STA BASE+$601,X
8035- 9D 01 27          0000>         STA BASE+$701,X
                        0000>         .DO BASE<$5800
8038-                   0000>         >ZERO BASE+$800
8038- 9D 01 28          0000>>         STA BASE+$800+$001,X
803B- 9D 01 29          0000>>         STA BASE+$800+$101,X
803E- 9D 01 2A          0000>>         STA BASE+$800+$201,X
8041- 9D 01 2B          0000>>         STA BASE+$800+$301,X
8044- 9D 01 2C          0000>>         STA BASE+$800+$401,X
8047- 9D 01 2D          0000>>         STA BASE+$800+$501,X
804A- 9D 01 2E          0000>>         STA BASE+$800+$601,X
804D- 9D 01 2F          0000>>         STA BASE+$800+$701,X
                        0000>>         .DO BASE+$800<$5800
8050-                   0000>>         >ZERO BASE+$800+$800
8050- 9D 01 30          0000>>>         STA BASE+$800+$800+$001,X
8053- 9D 01 31          0000>>>         STA BASE+$800+$800+$101,X
8056- 9D 01 32          0000>>>         STA BASE+$800+$800+$201,X
8059- 9D 01 33          0000>>>         STA BASE+$800+$800+$301,X
805C- 9D 01 34          0000>>>         STA BASE+$800+$800+$401,X
805F- 9D 01 35          0000>>>         STA BASE+$800+$800+$501,X
8062- 9D 01 36          0000>>>         STA BASE+$800+$800+$601,X
8065- 9D 01 37          0000>>>         STA BASE+$800+$800+$701,X
                        0000>>>         .DO BASE+$800+$800<$5800
8068-                   0000>>>         >ZERO BASE+$800+$800+$800
```

QUICKTRACE

relocatable program traces and displays the actual machine operations, *while* it is running without interfering with those operations. Look at these *FEATURES:*

***Single-Step** mode displays the last instruction, next instruction, registers, flags, stack contents, and six user-definable memory locations.*

***Trace** mode gives a running display of the Single-Step information and can be made to stop upon encountering any of nine user-definable conditions.*

***Background** mode permits tracing with no display until it is desired. Debugged routines run at near normal speed until one of the stopping conditions is met, which causes the program to return to Single-Step.*

***QUICKTRACE** allows changes to the stack, registers, stopping conditions, addresses to be displayed, and output destinations for all this information. All this can be done in Single-Step mode while running.*

***Two optional display formats** can show a sequence of operations at once. Usually, the information is given in four lines at the bottom of the screen.*

***QUICKTRACE** is completely transparent to the program being traced. It will not interfere with the stack, program, or I/O.*

***QUICKTRACE** is relocatable to any free part of memory. Its output can be sent to any slot or to the screen.*

***QUICKTRACE** is completely compatible with programs using Applesoft and Integer BASICs, graphics, and DOS. (Time dependent DOS operations can be bypassed.) It will display the graphics on the screen while **QUICKTRACE** is alive.*

***QUICKTRACE** is a beautiful way to show the incredibly complex sequence of operations that a computer goes through in executing a program*

See these programs at participating Computerland and other fine computer stores.

Anthro - Digital Software, Inc.
P.O. Box 1385 Pittsfield, MA 01202

QUICKTRACE *$50*

Written by John Rogers

```
8068- 9D 01 38 0000>>>>         STA BASE+$800+$800+$800+$001,X
806B- 9D 01 39 0000>>>>         STA BASE+$800+$800+$800+$101,X
806E- 9D 01 3A 0000>>>>         STA BASE+$800+$800+$800+$201,X
8071- 9D 01 3B 0000>>>>         STA BASE+$800+$800+$800+$301,X
8074- 9D 01 3C 0000>>>>         STA BASE+$800+$800+$800+$401,X
8077- 9D 01 3D 0000>>>>         STA BASE+$800+$800+$800+$501,X
807A- 9D 01 3E 0000>>>>         STA BASE+$800+$800+$800+$601,X
807D- 9D 01 3F 0000>>>>         STA BASE+$800+$800+$800+$701,X
               0000>>>>         .DO BASE+$800+$800+$800<$5800
8080-          0000>>>>         >ZERO BASE+$800+$800+$800+$800
8080- 9D 01 40 0000>>>>>         STA BASE+$800+$800+$800+$800+$001,X
8083- 9D 01 41 0000>>>>>         STA BASE+$800+$800+$800+$800+$101,X
8086- 9D 01 42 0000>>>>>         STA BASE+$800+$800+$800+$800+$201,X
8089- 9D 01 43 0000>>>>>         STA BASE+$800+$800+$800+$800+$301,X
808C- 9D 01 44 0000>>>>>         STA BASE+$800+$800+$800+$800+$401,X
808F- 9D 01 45 0000>>>>>         STA BASE+$800+$800+$800+$800+$501,X
8092- 9D 01 46 0000>>>>>         STA BASE+$800+$800+$800+$800+$601,X
8095- 9D 01 47 0000>>>>>         STA BASE+$800+$800+$800+$800+$701,X
               0000>>>>>         .DO BASE+$800+$800+$800+$800<$5800
8098-          0000>>>>>         >ZERO BASE+$800+$800+$800+$800+$800
8098- 9D 01 48 0000>>>>>>         STA BASE+$800+$800+$800+$800+$800+$001,X
809B- 9D 01 49 0000>>>>>>         STA BASE+$800+$800+$800+$800+$800+$101,X
809E- 9D 01 4A 0000>>>>>>         STA BASE+$800+$800+$800+$800+$800+$201,X
80A1- 9D 01 4B 0000>>>>>>         STA BASE+$800+$800+$800+$800+$800+$301,X
80A4- 9D 01 4C 0000>>>>>>         STA BASE+$800+$800+$800+$800+$800+$401,X
80A7- 9D 01 4D 0000>>>>>>         STA BASE+$800+$800+$800+$800+$800+$501,X
80AA- 9D 01 4E 0000>>>>>>         STA BASE+$800+$800+$800+$800+$800+$601,X
80AD- 9D 01 4F 0000>>>>>>         STA BASE+$800+$800+$800+$800+$800+$701,X
               0000>>>>>>         .DO BASE+$800+$800+$800+$800+$800<$5800
80B0-          0000>>>>>>         >ZERO BASE+$800+$800+$800+$800+$800+$800
80B0- 9D 01 50 0000>>>>>>>         STA BASE+$800+$800+$800+$800+$800+$800+$001,X
80B3- 9D 01 51 0000>>>>>>>         STA BASE+$800+$800+$800+$800+$800+$800+$101,X
80B6- 9D 01 52 0000>>>>>>>         STA BASE+$800+$800+$800+$800+$800+$800+$201,X
80B9- 9D 01 53 0000>>>>>>>         STA BASE+$800+$800+$800+$800+$800+$800+$301,X
80BC- 9D 01 54 0000>>>>>>>         STA BASE+$800+$800+$800+$800+$800+$800+$401,X
80BF- 9D 01 55 0000>>>>>>>         STA BASE+$800+$800+$800+$800+$800+$800+$501,X
80C2- 9D 01 56 0000>>>>>>>         STA BASE+$800+$800+$800+$800+$800+$800+$601,X
80C5- 9D 01 57 0000>>>>>>>         STA BASE+$800+$800+$800+$800+$800+$800+$701,X
               0000>>>>>>>         .DO BASE+$800+$800+$800+$800+$800+$800<$5800
80C8-          0000>>>>>>>         >ZERO BASE+$800+$800+$800+$800+$800+$800+$800
80C8- 9D 01 58 0000>>>>>>>>         STA BASE+$800+$800+$800+$800+$800+$800+$800+$001,X
80CB- 9D 01 59 0000>>>>>>>>         STA BASE+$800+$800+$800+$800+$800+$800+$800+$101,X
80CE- 9D 01 5A 0000>>>>>>>>         STA BASE+$800+$800+$800+$800+$800+$800+$800+$201,X
80D1- 9D 01 5B 0000>>>>>>>>         STA BASE+$800+$800+$800+$800+$800+$800+$800+$301,X
80D4- 9D 01 5C 0000>>>>>>>>         STA BASE+$800+$800+$800+$800+$800+$800+$800+$401,X
80D7- 9D 01 5D 0000>>>>>>>>         STA BASE+$800+$800+$800+$800+$800+$800+$800+$501,X
80DA- 9D 01 5E 0000>>>>>>>>         STA BASE+$800+$800+$800+$800+$800+$800+$800+$601,X
80DD- 9D 01 5F 0000>>>>>>>>         STA BASE+$800+$800+$800+$800+$800+$800+$800+$701,X
               0000>>>>>>>>         .DO BASE+$800+$800+$800+$800+$800+$800+$800<$5800
               0000>>>>>>>>         .FIN
               0000>>>>>>>        .FIN
               0000>>>>>>       .FIN
               0000>>>>>      .FIN
               0000>>>>     .FIN
               0000>>>    .FIN
               0000>>   .FIN
               0000>  .FIN
80E0- E8       1460         INX            EVERY ODD LOCATION
80E1- E8       1470         INX
80E2- F0 03    1480         BEQ .2
80E4- 4C 20 80 1490         JMP .1         NOT FINISHED CLEARING
               1500 *--------------------------------
80E7- A9 03    1510 .2      LDA #3
80E9- 8D 27 81 1520         STA START+1
80EC- 0A       1530 MAINLP  ASL            INC = START * 2
80ED- 8D 1A 81 1540         STA INC+1
80F0- A9 00    1550 SQUARE  LDA #*-*       MOVE MULT TO N
80F2- 8D 18 81 1560         STA N+2
80F5- AD 33 81 1570         LDA MULT+1
80F8- 0A       1580         ASL            MULTIPLY BY 8
80F9- 2E 18 81 1590         ROL N+2
80FC- 0A       1600         ASL
80FD- 2E 18 81 1610         ROL N+2
8100- 0A       1620         ASL
8101- 2E 18 81 1630         ROL N+2
8104- AA       1640         TAX
8105- E8       1650         INX            AND ADD 1
8106- D0 03    1660         BNE .1
8108- EE 18 81 1670         INC N+2
```

```
810B- 18        1680 .1     CLC           ADD BASE TO N
810C- AD 18 81  1690        LDA N+2
810F- 69 20     1700        ADC /BASE
8111- 8D 18 81  1710        STA N+2
8114- A8        1720        TAY
8115- 8A        1730        TXA
                1740 LOOP
8116- 9D 00 FF  1750 N      STA $FF00,X   REMEMBER THAT N IS REALLY AT N+2
8119- 69 00     1760 INC    ADC #*-*      N = N + INC
811B- AA        1770        TAX
811C- 90 F8     1780        BCC LOOP      DONT'T BOTHER TO ADD, NO CARRY
811E- C8        1790        INY           INC HIGH ORDER
811F- 8C 18 81  1800        STY N+2
8122- C0 60     1810        CPY /BASE+$4000  IF IS GREATER THAN $6000
8124- 90 F0     1820        BCC LOOP      NO, REPEAT
8126- A2 00     1830 START  LDX #*-*      GET OUR NEXT KNOCKOUT
8128- E8        1840 NEXT   INX
8129- E8        1850        INX           START = START + 2
812A- 30 1B     1860        BMI END       WE'RE DONE IF X>$7F
812C- EE 30 81  1870        INC SHCNT+1   INCREMENT SQUARE MULTIPLIER
812F- A9 00     1880 SHCNT  LDA #*-*      AND ADD TO MULTIPLIER
8131- 18        1890        CLC
8132- 69 00     1900 MULT   ADC #*-*
8134- 8D 33 81  1910        STA MULT+1
8137- 90 03     1920        BCC .1
8139- EE F1 80  1930        INC SQUARE+1
813C- BD 00 20  1940 .1     LDA BASE,X    GET A POSSIBLE PRIME
813F- D0 E7     1950        BNE NEXT      THIS ONE HAS BEEN KNOCKED OUT
8141- 8E 27 81  1960        STX START+1
8144- 8A        1970        TXA
8145- D0 A5     1980        BNE MAINLP    ...ALWAYS
8147- 60        1990 END    RTS
                2000 *--------------------------------
8148- 00        2010 COUNT  .DA #*-*       COUNT FOR 100 TIMES LOOP
```

Moving the Symbol Table .....Bill Morgan

Do you use the language card version of the S-C Macro Assembler? Have you ever tried to create more space for your object code by patching $D01D to move the symbol table up from $1000? Got a MEM PROTECT ERROR, didn't you? Here's what went wrong, and how to fix it.

The problem is the private label table for macros. This table is also protected during assembly, and starts at $FFF and grows downward. The base of the table is defined by a LDA #$10 instruction at $E564. When the table is searched during assembly, the check for the end of the table is a CMP #$10 at $E6A0. Both of these must also be patched to allow the $D01D patch to work. Here are the commands to correct the assembler:

```
:$C083 C083 N E564:A5 4B N E6A0:C5 4B N C080
:BSAVE S-C.ASM.MACRO.LC,A$D000,L$231F
```

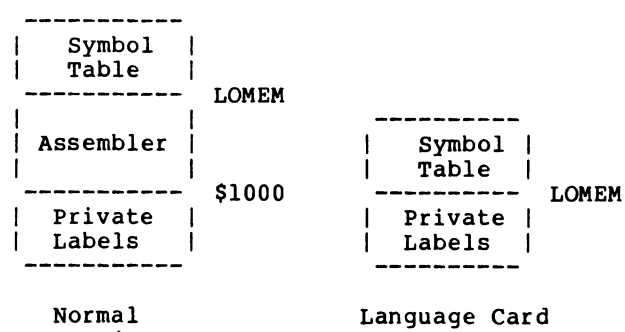
This changes the LDA #$10 to a LDA LOMEM+1 and the CMP #$10 to a CMP LOMEM+1. Now, whenever you want to move the symbol table, just type the following (where XX is the page you want the tables to start with):

```
:$C083 C083 D01D:XX N C080
:NEW
```

The NEW command is necessary to reset the page-zero pointers.

If you are using a target file and don't care about object code space, you can move the symbol table down. This creates more source code and symbol table space. You can move the table base all the way down to $800, if you are not using private labels. If you are using them, remember that each private label occurence uses 5 bytes of table space, so be sure to leave enough room under the table base.

Here's a map that shows how things got this way:



The normal version of the assembler has to start at $1000, so the private label table also has to be there. The language card version didn't get changed to reflect the fact that the private labels could now be moved.

EXEC without END from Applesoft.....Bob Sander-Cederlof

I have been working on a project with Lee Meador which requires a binary file to be loaded into the second $D000 bank of a 16K RAM card. It is just a little tricky to do this!

You cannot just use a simple BLOAD, because you have to be sure the RAM card is selected and write-enabled. You cannot do it from a running Applesoft program, or even as a direct command after the Applesoft prompt, because if the RAM card is enabled the Applesoft ROMs are not. We wanted to do it from within the running Applesoft program.

The typical answer is to create an EXEC file with the commands to call the monitor, select the RAM card, BLOAD the file, reselect the motherboard ROMs, and bounce back to Applesoft. For example:

```
CALL-151              call Apple monitor
C089 C089             write-enable RAM with 2nd bank
F800<F800.FFFFM       copy of monitor in RAM card
BLOAD B.BOBANDLEE     load the file
C081                  back to Applesoft ROMs
3D0G                  back to Applesoft, softly
```

You can nicely EXEC this file from the direct mode, or from a running Applesoft program. However, in order to use it from a running program, the program must END or STOP. Do it like this:

```
100 PRINT CHR$(4)"EXEC LOAD 2ND BANK":END
```

If you don't END the program, the EXEC file will probably just become part of the input to your Applesoft program, rather than being executed.

HOWEVER.... You can beat the system. Change the EXEC file to this form:

```
C089 C089
F800<F800.FFFFM
BLOAD B.BOBANDLEE
C081
D7D2G
```

And the Applesoft code to this:

```
100 PRINT CHR$(4)"EXEC LOAD 2ND BANK":CALL-151
```

Note the two changes in the EXEC file: the CALL-151 is not there, and 3D0G has become D7D2G. And in the Applesoft code instead of END we have CALL-151.

The CALL-151 starts up the Apple monitor, which reads the commands from the EXEC file. The last command jumps to $D7D2, the running entry into Applesoft. This continues execution of the Applesoft program from the next statement after the CALL-151.

Applesoft Program Locator.....Bill Morgan

Have you ever wanted to know exactly where your Applesoft program and variables are in memory? How much space is code and how much is variables? How close you're getting to the Hi-Res display space? FRE(0) will tell you how much space you have, but not where it is. You can PEEK the Applesoft pointers, or go into the monitor to check them, but that means you have to remember where all the pointers are.

In the October, 1982 issue of Big Apple Users Digest I saw a program by Frank Weinberg to build an EXEC file called FPSTAT, which displays the Applesoft pointers to program and variable locations. (That program was credited as being reprinted from The Grapevine, August, 1982.) Now that was pretty neat, but EXEC is so slow, and the adresses were printed in decimal. I'm more comfortable thinking of addresses in hex notation. Bob suggested writing a BRUNnable program which would execute in page 2 (the input buffer), thus avoiding conflict with any page 3 routines that might be present. Here's what I came up with.

Using LOCATOR

Whenever you want to know the memory situation, just BRUN LOCATOR. It will display something like this:

```
PROGRAM: $0801 TO $0923
 SIMPLE: $0923 TO $0A35
 ARRAYS: $0A35 TO $1B3C
STRINGS: $9435 TO $9600
```

PROGRAM shows the location of the actual text of your program. SIMPLE is the simple variables, both numeric and string pointers. ARRAYS is the array variables, both numeric and string. STRINGS is the area used by the actual text of the strings.

Notice that the upper addresses are all one too large. Applesoft's end-of-program and end-of-variables pointers actually point to the next available location, rather than the last location used. Similarly, the end-of-strings pointer is HIMEM, which is one past the last location available. I wrote another version of LOCATOR which automatically decremented the second address in each line, but that got cumbersome, and returned silly values if the Applesoft program had not yet been RUN. (For example, SIMPLE: $0923 TO $0922.)

If you want to CALL LOCATOR from within an Applesoft program, change line 1320 from JMP $3D0 to RTS, and change the origin to $294. Then you can CALL 660, if you're not using very long input lines. Or, you can put LOCATOR in page 3, if you're not already using that area.

It is also interesting to RUN a program, BRUN LOCATOR, then type FRE(0) and call LOCATOR again. This lets you see just how much wasted string space you have had, and gives you some idea how long the garbage collector takes to clear how much space.

I'm looking forward to using LOCATOR together with EXAMINER (from AAL June, 1982) to study Applesoft's variable structure. You can find more information on Applesoft variables and their pointers on pages 126-127 and 137 of the Applesoft manual.

How LOCATOR Works

Since we are printing eight addresses, the X-register is used to count from 0-7. In lines 1140-1190 that count is converted into a value of $0, $8, $10, or $18, to determine which title line to print. If the titles hadn't been a convenient 8 bytes long, we could have inserted a title offset at the beginning of each of the .DA statements in lines 1570-1600, and loaded Y from there.

The heart of the program is the table of Applesoft pointers at lines 1570-1600. In lines 1420-1440 the Y-register is loaded with a value from the table, then used to load the A- and X-registers with the address pointed to. The program then calls MON.PRNTAX, which displays first the A- and then the X-register.

```
                    1010 *--------------------------------
                    1020         .OR $292     HIGH END OF INPUT BUFFER
                    1030 *       .TF LOCATOR
                    1040 *--------------------------------
0000-               1050 ZERO        .EQ 0
                    1060
F941-               1070 MON.PRNTAX .EQ $F941
FDED-               1080 MON.COUT   .EQ $FDED
FD8E-               1090 MON.CROUT  .EQ $FD8E
                    1100 *--------------------------------
0292- A2 00         1110 START   LDX #0
                    1120
0294- 20 8E FD      1130 LOOP    JSR MON.CROUT      NEW LINE
0297- 8A            1140         TXA
0298- 4A            1150         LSR                MAKE (X)
0299- 0A            1160         ASL                INTO
029A- 0A            1170         ASL                TITLE
029B- 0A            1180         ASL                INDEX
029C- A8            1190         TAY
029D- B9 D3 02      1200 .1      LDA TITLES,Y       SHOW TITLE
02A0- 20 ED FD      1210         JSR MON.COUT
02A3- C8            1220         INY
02A4- C9 BA         1230         CMP #':+$80        ":" ?
02A6- D0 F5         1240         BNE .1
                    1250
02A8- A0 01         1260         LDY #1             FILL WITH " $"
02AA- 20 B9 02      1270         JSR PRINT.ADDRESS
02AD- A0 04         1280         LDY #4             FILL WITH " TO $"
02AF- 20 B9 02      1290         JSR PRINT.ADDRESS
02B2- E0 07         1300         CPX #7             DONE YET?
02B4- 90 DE         1310         BCC LOOP           NO, GO ON
02B6- 4C D0 03      1320         JMP $3D0           YES, EXIT TO DOS
                    1330 *--------------------------------
                    1340 PRINT.ADDRESS
02B9- B9 FB 02      1350 .1      LDA FILLER,Y       Y TELLS HOW
02BC- 20 ED FD      1360         JSR MON.COUT       MUCH FILLER
02BF- 88            1370         DEY                TO PRINT
02C0- 10 F7         1380         BPL .1
                    1390
02C2- 8A            1400         TXA
02C3- 48            1410         PHA                SAVE X
02C4- BC F3 02      1420         LDY TABLE,X        GET POINTER
02C7- B9 01 00      1430         LDA ZERO+1,Y       GET HIGH BYTE
02CA- B6 00         1440         LDX ZERO,Y         GET LOW BYTE
02CC- 20 41 F9      1450         JSR MON.PRNTAX     DISPLAY ADDRESS
02CF- 68            1460         PLA
02D0- AA            1470         TAX                RESTORE X
02D1- E8            1480         INX                AND GET READY
02D2- 60            1490         RTS                FOR NEXT PASS
                    1500 *--------------------------------
                    1510 TITLES
02D3- D0 D2 CF
02D6- C7 D2 C1
02D9- CD BA         1520         .AS -/PROGRAM:/
02DB- A0 D3 C9
02DE- CD D0 CC
02E1- C5 BA         1530         .AS -/ SIMPLE:/
02E3- A0 C1 D2
02E6- D2 C1 D9
02E9- D3 BA         1540         .AS -/ ARRAYS:/
02EB- D3 D4 D2
02EE- C9 CE C7
02F1- D3 BA         1550         .AS -/STRINGS:/
                    1560 *--------------------------------
02F3- 67 AF         1570 TABLE   .DA #$67,#$AF      START OF PROGRAM, END OF PROGRAM
02F5- 69 6B         1580         .DA #$69,#$6B      START OF VARIABLES, START OF ARRAYS
02F7- 6B 6D         1590         .DA #$6B,#$6D      START OF ARRAYS, END OF NUMERICS
02F9- 6F 73         1600         .DA #$6F,#$73      START OF STRINGS, HIMEM
                    1610 *--------------------------------
02FB- A4 A0 CF
02FE- D4 A0         1620 FILLER .AS -/$ OT /
```

HardCore Computing is a quarterly magazine.

REPEAT and UNTIL for Applesoft.....Bobby Deen

The following program adds three statements to Applesoft: &REPEAT, &UNTIL, and &POPR. With these you can write Pascal-like loops in your Basic programs.

You start the loop with &REPEAT, and end it with &UNTIL <exp>. The loop will be repeated until the <exp> evaluates to non-zero (true). As long as the value of <exp> is zero (false), the loop will keep going.

I use the system stack for saving the line number and the program pointer, just like Applesoft does with FOR-NEXT loops. A special code is used to identify the stuff on the stack, so you can have FOR-NEXT loops inside REPEAT-UNTIL loops and vice versa.

The statement &POPR removes one REPEAT block from the stack, in case you want to jump out of a loop rather than completing it. (This is not generally a good practice, even with FOR-NEXT loops, but you can do it if you feel you must.) The statement "&UNTIL 1" will do the same thing as &POPR, but &POPR takes less space and time.

If &POPR or &UNTIL is executed when there is not an UNTIL block on the top of the stack, you will get "NEXT WITHOUT FOR" error.

Applesoft parses the word "REPEAT" as four letters "REPE" and the token "AT". This makes the listings look weird, but never mind. Likewise, "UNTIL" looks like a variable name during tokenization, so the expression runs into the letter "L"; but at execution time all is understood.

Here is a sample program which shows a pair of REPEAT loops:

```
100  REM TEST REPEAT/UNTIL
110 D$ =  CHR$ (4): PRINT D$"BLOAD B.REPEAT/UNTIL": CALL 768
120 I = 0: & REPE AT
130 I = I + 1: PRINT I":  ";
135 J = 0: & REPE AT :J = J + 1: PRINT J" ";: & UNTILJ > 14:
    PRINT
140  & UNTILI = 10
```

Lines 1200-1250 install the ampersand vector. I assumed the JMP opcode is already stored at $3F5, since DOS does that. After BLOADing the file, CALL 768 will executed these lines.

When the "&" is executed, the 6502 jumps to AMPER.PARSE at line 1270. Lines 1270-1420 search through a table of keywords, matching one if possible with the characters after the "&" in your Applesoft program. This is a general routine, which you can use for any keywords, just by making the appropriate entries in the table (lines 1500-1590).

The table contains a string and an address for each keyword. The string is shown as a hex string, and includes the exact hexadecimal values expected. For example, for "REPEAT" I have entered the ASCII codes for "REPE" and the token value or "AT".

After the keyword there is a 00 byte to flag the end, and a two byte address. The address will be pushed onto the stack so that an RTS instruction will branch to the processing program for that keyword. Since RTS adds one, the address in the table have "-1" after them.

The last entry in the table has a null keyword, so it will match anything and everything. If the search goes this far, we have a syntax error; therefore the branch address is to the Applesoft syntax error code.

When a keyword is matched, the Y-register contents need to be added to TXTPTR. A subroutine in the Applesoft ROMs does this, called AS.ADDON. Since both REPEAT and POPR require the next character to be end-of-line or a colon, a JMP to AS.CHRGOT gets the next character and tests it. The RTS at the end of AS.CHRGOT actually branches to the processing code for the keyword.

Lines 1600-1840 process the REPEAT command. A five-byte block is pushed onto the stack, consisting of the current line number, the TXTPTR, and a code value $B8.

Lines 1850-2070 process the UNTIL command. First the expression is evaluated. If the value turns out to be zero, the byte at FAC.EXP will be zero. If it is zero, we need to keep looping; if non-zero, the loop is finished. Looping involves copying the line number and text pointer from the stack back into CURLIN and TXTPTR, and then going to AS.NEWSTT. The REPEAT block is left on the stack, and execution resumes just after the &REPEAT that started this loop.

If the expression is true (non-zero), the loop is terminated. Termination is trivial: just pop off the REPEAT block, and go to AS.NEWSTT to continue execution after the UNTIL statement. I could pop the block off with seven PLA's, but I used the technique of adding 7 to the stack pointer instead.

Naturally, this package was assembled to sit in page 3, along with 99 other machine language things you use. You can easily move it to another location, just by changing the origin (line 1180). Or you can use the routines with Amper-Magic or the Routine Machine. Note that the routines themselves are relocatable run-anywhere code (no data references, JSR's, or JMP's to points within the routines). You will have to shorten the routine names to four or less characters to use them with Amper-Magic.

Pascal has some other looping constructs which you might like to see in Applesoft. Now that you see how I did this one, why not try your hand at coding REPEAT WHILE?

```
                    1010 *--------------------------------
                    1020 *   BY BOBBY DEEN
                    1030 *      629 WINCHESTER DR
                    1040 *      RICHARDSON,TX. 75080
                    1050 *      (214) 235-4391
                    1060 *--------------------------------
03F5-               1070 AMPERSAND.VECTOR .EQ $3F5
DD7B-               1080 AS.FRMEVL   .EQ $DD7B      EVALUATE A FORMULA
00B7-               1090 AS.CHRGOT   .EQ $00B7      GET CHAR AT TXTPTR
00B8-               1100 AS.TXTPTR   .EQ $00B8      POINT TO PROGRAM TEXT
DEC9-               1110 AS.SYNERR   .EQ $DEC9      SYNTAX ERROR
D998-               1120 AS.ADDON    .EQ $D998      ADDS (Y) TO TXTPTR
0075-               1130 AS.CURLIN   .EQ $75        CURRENT LINE NUMBER
009D-               1140 FAC.EXP     .EQ $9D        EXPONENT OF FAC
DD0B-               1150 AS.BADFOR   .EQ $DD0B      NEXT WITHOUT FOR ERROR
D7D2-               1160 AS.NEWSTT   .EQ $D7D2      EXECUTE NEW STATEMENT
                    1170 *--------------------------------
                    1180         .OR $300
                    1190         .TF B.REPEAT/UNTIL
                    1200 *--------------------------------
0300- A9 0B         1210 START   LDA #AMPER.PARSE
0302- 8D F6 03      1220         STA AMPERSAND.VECTOR+1
0305- A9 03         1230         LDA /AMPER.PARSE
0307- 8D F7 03      1240         STA AMPERSAND.VECTOR+2
030A- 60            1250         RTS
                    1260 *--------------------------------
                    1270 AMPER.PARSE
030B- A2 FF         1280         LDX #-1        START OF TABLE
030D- A0 FF         1290 .1      LDY #-1        START OF AMPER-CALL
030F- E8            1300 .2      INX
0310- C8            1310         INY
0311- BD 32 03      1320         LDA TABLE,X    NEXT CHAR FROM TABLE
0314- F0 0E         1330         BEQ .4         END OF KEYWORD, MATCHED
0316- D1 B8         1340         CMP (AS.TXTPTR),Y   COMPARE WITH AMPER-CALL
0318- F0 F5         1350         BEQ .2         MATCHES SO FAR
                    1360 *---SKIP TO NEXT TABLE ENTRY-----
031A- E8            1370 .3      INX            ...TO END OF KEYWORD
031B- BD 32 03      1380         LDA TABLE,X
031E- D0 FA         1390         BNE .3
0320- E8            1400         INX            ...OVER THE ADDRESS
0321- E8            1410         INX
0322- D0 E9         1420         BNE .1         ...ALWAYS
                    1430 *---MATCHED A KEYWORD------------
0324- 20 98 D9      1440 .4      JSR AS.ADDON       ADJUST TXTPTR PAST KEYWORD
0327- BD 34 03      1450         LDA TABLE+2,X      GET ADDRESS AND BRANCH
032A- 48            1460         PHA
032B- BD 33 03      1470         LDA TABLE+1,X
032E- 48            1480         PHA
032F- 4C B7 00      1490         JMP AS.CHRGOT      GET CHAR AT TXTPTR
                    1500 *--------------------------------
                    1510 TABLE
0332- 52 45 50
0335- 45 C5 00      1520         .HS 52455045C500     "REPEAT"
0338- 49 03         1530         .DA REPEAT-1
033A- 55 4E 54
033D- 49 4C 00      1540         .HS 554E54494C00     "UNTIL"
0340- 63 03         1550         .DA UNTIL-1
0342- A1 52 00      1560         .HS A15200           "POPR"
0345- 8C 03         1570         .DA POPR-1
0347- 00            1580         .HS 00               ANYTHING
0348- C8 DE         1590         .DA AS.SYNERR-1
                    1600 *--------------------------------
                    1610 * REPEAT COMMAND
                    1620 *--------------------------------
                    1630 REPEAT
034A- D0 57         1640         BNE SYNERR       NOT THERE
034C- 68            1650         PLA            SAVE RETURN ADDRESS
034D- AA            1660         TAX
034E- 68            1670         PLA
034F- A8            1680         TAY
0350- A5 76         1690         LDA AS.CURLIN+1 PUSH CURRENT LINE NUMBER
0352- 48            1700         PHA
0353- A5 75         1710         LDA AS.CURLIN
0355- 48            1720         PHA
0356- A5 B9         1730         LDA AS.TXTPTR+1 PUSH TEXT POINTER
0358- 48            1740         PHA
0359- A5 B8         1750         LDA AS.TXTPTR
035B- 48            1760         PHA
```

APPLE PERIPHERALS ARE OUR ONLY BUSINESS

TIME II

THE MOST POWERFUL, EASIEST TO USE CLOCK FOR YOUR APPLE

- Time in hours, minutes and seconds.
- Date with year, month, day of week and leap year.
- Will enhance programs for accounting, time and energy management, remote control of appliances, laboratory analysis, process control, and more.
- 24-hour military format or 12-hour with AM/PM indication.
- User selectable interrupts permit foreground/background operati of two programs simultaneously.
- Crystal controlled for .0005% accuracy.
- Easy programming in basic.
- On board battery backup power for over four operation (battery charges when Apple is on).

- Twenty-seven page operating manual included with many examples of programs to use with your Apple in any configuration.
- Includes disk containing a DOS Dater and many other time oriented utilities plus over 25 user contributed programs at no extra cost.

PRICE $129.00

SUPER MUSIC SYNTHESIZER

- Complete 16 voice music synthesizer on one card. Just plug it into your Apple, connect the audio cable (supplied) to your stereo and boot the disk supplied and you are ready to input and play songs.
- It's easy to program music with our compose software. You will start right away at inputting your favorite songs. The Hi-Res screen shows what you have entered in standard sheet music format.

- We give you lots of software. In addition to Compose and Play programs, the disk is filled with songs ready to run.
- Easy to program in basic to generate complex sound effects.
- Four white noise generators which are great for sound effects.
- Plays music in true stereo as well as true discrete quadraphonic.
- Envelope control.
- Will play songs written for ALF synthesizer (ALF software will not take advantage of all the features of this board. Their software sounds the same in our synthesizer.)
- Automatic shutoff on power-up or if reset is pushed.
- Many many more features.

PRICE $159.00

ANALOG TO DIGITAL CONVERTER

- 8 Channels
- 8 Bit Resolution
- On Board Memory
- Ratiometric Capability
- Fast Conversion (.078 ms per channel)
- Eliminates The Need To Wait For A/D Conversion (just PEEK at data)
- A/D Process Totally Transparent to Apple (looks like memory)

The analog to digital conversion takes place on a continuous, channel sequencing basis. Data is automatically transferred to on board memory at the end of each conversion. No A/D converter could be easier to use.

Our A/D board comes standard with 0, 10V full scale inputs. These inputs can be changed by the user to 0, -10V, or -5V, +5V or other ranges as needed.

The user connector has + 12 and -12 volts on it so you can power your sensors. (These power sources can be turned off with on board dip switch).

Accuracy 0.3% Input Resistance 20K Ohms Typ

A few applications may include the monitoring of ● flow ● temperature ● humidity ● wind speed ● wind direction ● light intensity ● pressure ● RPM ● soil moisture and many more.

PRICE $129.00

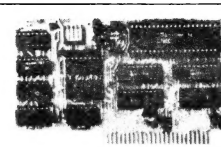
DIGITAL INPUT/OUTPUT BOARD

- Provides 8 buffered outputs to a standard 16 pin socket for standard dip ribbon cable connection.
- Power-up reset assures that all outputs are off when your Apple i first turned on.
- Features 8 inputs that can be driven from TTL logic or any 5 volt source.
- Your inputs can be anything from high speed logic to simple switches.
- Very simple to program, just PEEK at the data.
- 4 other outputs are also provided. User 1, reset, interrupt request, non-maskable interrupt.
- Now on one card, you can have 8 digital outputs and 8 digital inputs each with its own connector. The super input/output board is your best choice for any control application.

PRICE $62.00

Z-80 CARD

- TOTALLY compatible with all CP/M software.
- Executes the full Z-80 and 8080 instruction set.
- Allows you to run your Apple CP/M based programs.
- Does EVERYTHING the other Z-80 boards do, plus supports Z80 Interrupts.
- Hardware and software settable switch options.
- An on-card PROM eli inates many I.C.'s for a cooler, less power consuming board.
- Complete documentation included. (user must furnish software)



PRICE $139.00

Since our inception, Applied Engineering has continually expanded its line of Apple peripherals bringing you easy to-use designs.

We are the innovators not the imitators. Utilizing state-of the-art technologies, Applied Engineering is continually improving its products. The above represents our most recent development. Applied Engineering offers you the highest quality peripherals at the lowest possible price.

Applied Engineering's products are fully tested with complete documentation and available for immediate delivery. All products are guaranteed with a 9 warranty.

All Orders Shipped Same Day.
Texas Residents Add 5% Sales Tax.
Add $10.00 if Outside U.S.A.

Send Check or Money Order to
APPLIED ENGINEERING
P.O. Box 470301
Dallas, TX 75247

See Your Dealer
or Call (214) 492-2027
7 Days a Week
Master Card & Visa Welcome

```
035C- A9 B8     1770         LDA #$B8 IDENTIFIER FOR REPEAT LOOP
035E- 48        1780         PHA      SO THIS ISN'T MISTAKEN FOR FOR/NEXT
                1790 *               OR GOSUB/RETURN
035F- 98        1800         TYA       PUT RETURN ADDRESS ON STACK
0360- 48        1810         PHA
0361- 8A        1820         TXA
0362- 48        1830         PHA
0363- 60        1840         RTS       AND GO BACK
                1850 *-------------------------------
                1860 * PROCESS UNTIL COMMAND
                1870 *-------------------------------
                1880 UNTIL
0364- 20 7B DD  1890         JSR AS.FRMEVL GET EXPRESSION
0367- A5 9D     1900         LDA FAC.EXP  GET EXPONENT
0369- D0 24     1910         BNE POP.IT   TRUE,END LOOP
036B- BA        1920         TSX          KEEP LOOPING
036C- BD 03 01  1930         LDA $103,X
036F- C9 B8     1940         CMP #$B8     IS IT A REPEAT?
0371- D0 2D     1950         BNE BADFOR   NO,ERROR
0373- BD 04 01  1960         LDA $104,X   GET THE DATA
0376- 85 B8     1970         STA AS.TXTPTR AND TELL APPLESOFT
0378- BD 05 01  1980         LDA $105,X
037B- 85 B9     1990         STA AS.TXTPTR+1
037D- BD 06 01  2000         LDA $106,X
0380- 85 75     2010         STA AS.CURLIN
0382- BD 07 01  2020         LDA $107,X
0385- 85 76     2030         STA AS.CURLIN+1
0387- E8        2040         INX          WE DON'T NEED THE RETURN ADDRESS
0388- E8        2050         INX
0389- 9A        2060         TXS          KILL SUB CALL
038A- 4C D2 D7  2070         JMP AS.NEWSTT  NEW STATEMENT
                2080 *-------------------------------
                2090 * POP A REPEAT LOOP OFF STACK
                2100 *-------------------------------
                2110 POPR
038D- D0 14     2120         BNE SYNERR
038F- BA        2130 POP.IT  TSX          EXP TRUE,SO END LOOP
0390- BD 03 01  2140         LDA $103,X   MAKE SURE IT IS A REPEAT
0393- C9 B8     2150         CMP #$B8
0395- D0 09     2160         BNE BADFOR
0397- 8A        2170         TXA
0398- 18        2180         CLC
0399- 69 07     2190         ADC #7       PULL 7 THINGS
039B- AA        2200         TAX
039C- 9A        2210         TXS
039D- 4C D2 D7  2220         JMP AS.NEWSTT
                2230 *-------------------------------
03A0- 4C 0B DD  2240 BADFOR  JMP AS.BADFOR
03A3- 4C C9 DE  2250 SYNERR  JMP AS.SYNERR
                2260 *-------------------------------
```

Advertising in AAL

Once again, the price per page of advertising in Apple Assembly Line is going up. The December issue will run $90 for a full page, $50 for a half page.

Apple Assembly Line is published monthly by S-C SOFTWARE CORPORATION, P.O. Box 280300, Dallas, Texas 75228. Phone (214) 324-2050. Subscription rate is $15 per year in the USA, sent Bulk Mail; add $3 for First Class postage in USA, Canada, and Mexico; add $13 postage for other countries. Back issues are available for $1.50 each (other countries add $1 per back issue for postage).